Neuronale Netze

Hopfield Networks and Boltzmann Machines

Christian Mohr

22.11.2011

Institut für Anthropomatik, Interactive Systems Lab

Artificial Neural Networks

- Based on a simplified model of biological neural nets
  - Network of interconnected simple processing units (neurons)
    - Summarize activity of preceding units
    - “Fire” if sum exceeds a certain threshold
    - Firing is seen by subsequent units as activity

Artificial Neural Networks

Weights

- Influence of neurons on each other differs
  - Absolute value
  - Positive or negative (excitatory/inhibitory)
- Modeled as weights in ANNs
- Knowledge of neural nets is within the weights
- Learning means changing the weights

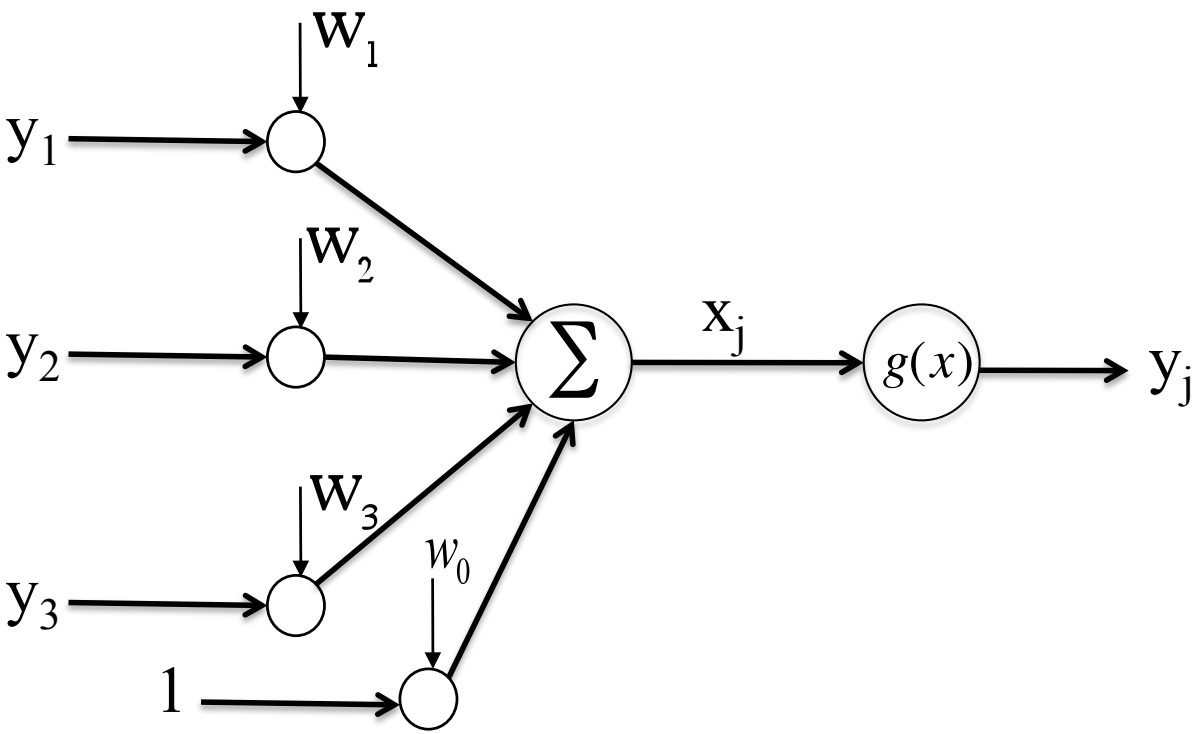
Artificial Neural Networks

Types of neurons

- 3 Types of neurons in a network:
  - Input Units
  - Hidden Units
  - Output Units
- Hidden Units can’t interact with the outside world (internal representation units)

Artificial Neural Networks

Connectionist Units



$$y_j = g(x_j) = g\left(\sum_i w_{ij} y_i\right)$$

Decision Function g(x)

$g(\vec{x}) > 0 \quad \Rightarrow$ Class A

$g(\vec{x}) < 0 \quad \Rightarrow$ Not class A

$g(\vec{x}) = 0 \quad \Rightarrow$ No decision

$$g(\vec{x}) = \sum_{i=1}^{n} w_1 x_1 + w_0 = \vec{w}^T \vec{x} + w_0 = \vec{w} \cdot \vec{x} + w_0$$

$\vec{x} = (x_1, .., x_n)^T$ Feature vector

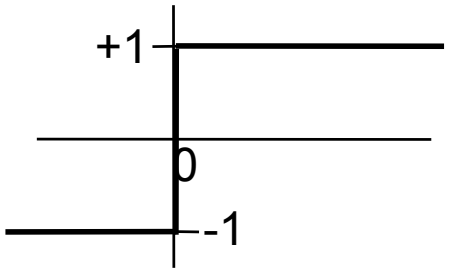
$\vec{w} = (w_1, ..., w_n)^T$ Weight vector

w_0 Threshold weight

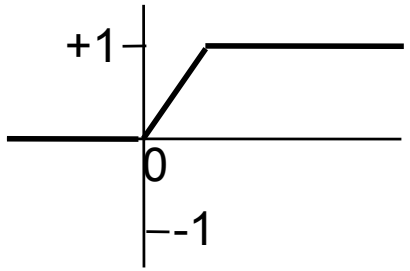
$\cdot$ denotes scalar product

Decision Function g(x)

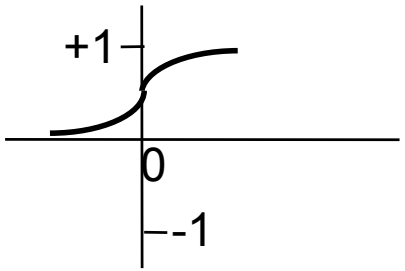
Three common non-linearities $f(*)$:



Hard Limiter



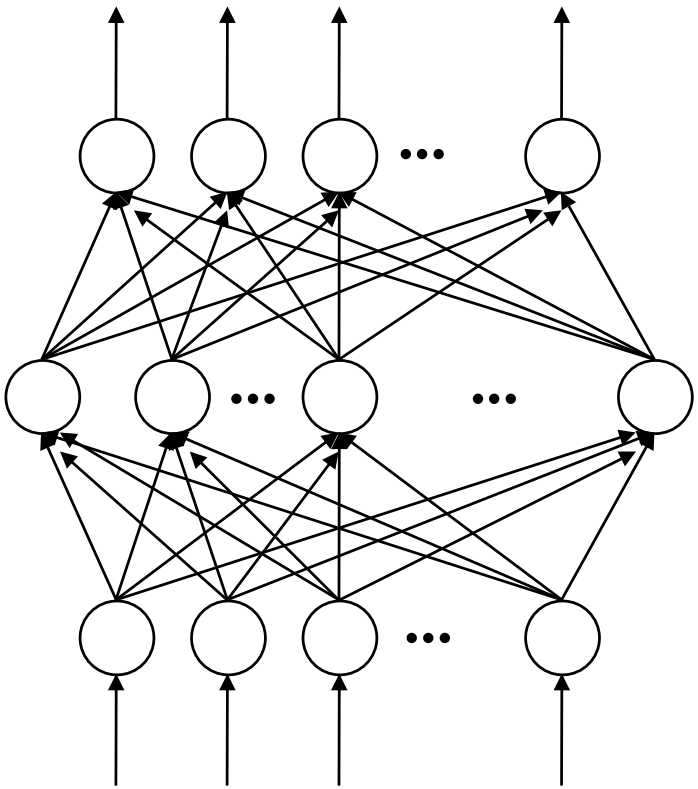
Threshold Logic



sigmoid $y_j = \frac{1}{1 + e^{-x_j}}$

Networks of Neurons/ Multi-Layer Perceptron

- Many interconnected simple processing elements:



Output patterns

Internal Representation Units

Input Patterns

Artificial Neural Networks

Learning Principles

- Depends on the actual type of network
- Common Principles:
  - Hebb Rule (biological adequate)
  - Delta Rule
  - Back Propagation (for Hidden Units)
  - Competitive Learning (unsupervised Learning)
  - …

Hopfield Nets

- Literatur:
  - Introduction to The Theory of Neural Computation
    Hertz, Krogh, Palmer, Santa Fe Institute
  - Neural Network Architectures – An Introduction,
    Judith Dayhoff, VNR Publishers

Hopfield Nets

- Introduced by John Hopfield in 1982
- Not very efficient but good to show principle of neural nets
- Corresponds to statistical mechanics (dynamics of magnets)
- Possible applications
  - Associative memory
  - Optimization

Binary Hopfield Nets

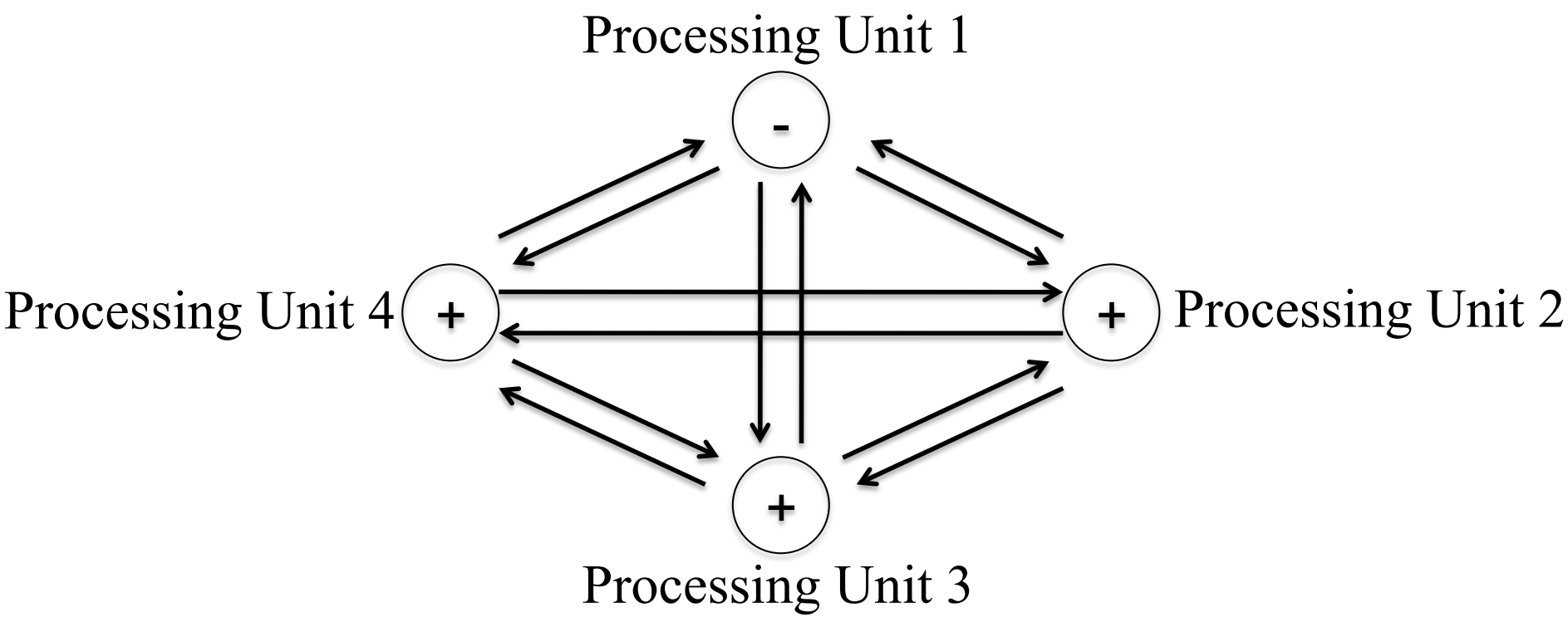
Basic Structure

- Single layer of processing units
- Each unit i has an activity value or “state” u_i
  - Binary: 0 or 1 (alternatively -1 or 1)
  - Denoted as + and - respectively
- Vector of unit states is the networks’ state

$$U = (u_1, u_2, \ldots, u_n) = (+,+,-,\ldots,+)$$

Binary Hopfield Nets

Example



Network State: $\vec{u} = (-,+,+,+)$

Binary Hopfield Nets

Connections

- Processing units fully interconnected
- Recurrent network topology
- Network can relax into stable states (without external input)
- Weights from unit j to unit i is T_{ji}
- Hopfield Nets: Weights between a pair of units are symmetric

$$T_{ji} = T_{ij}$$

Binary Hopfield Nets

Convergence

- Symmetric weights lead to the fact that the network will converge (relax in stable state)
- Some networks with $T_{ji} \neq T_{ij}$ can converge
- Convergence is condition for the network to perform useful computational tasks
- Weights are set in the beginning but the method depends on the application

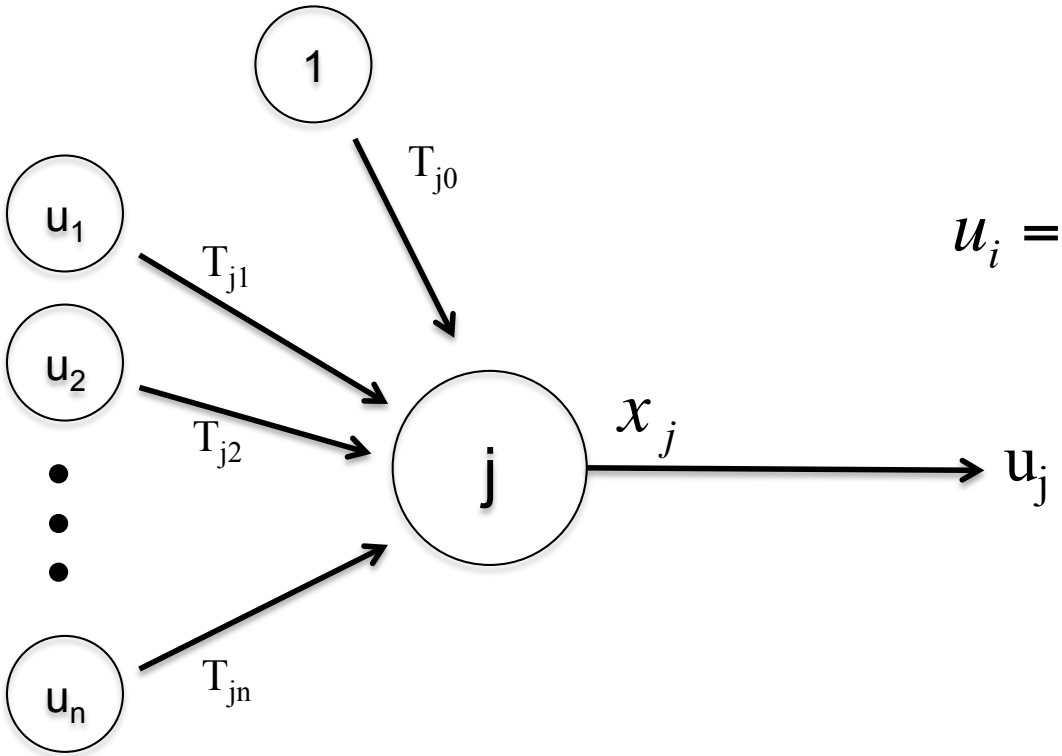
Binary Hopfield Nets

Updating Procedure

- Network state is initialized in the beginning
- Update one unit at a time
- Updating effects the state of the unit depending on the states of the remaining units and their weights to the unit being updated
- Continue updating until the network state does not change anymore

Binary Hopfield Nets

Processing Units



$$x_j = \sum_{\substack{i \\ i \neq j}} u_i T_{ji}$$

$$u_i = g(x_j) = \begin{cases} 1 & \text{if } x \geq 0; \\ 0 & \text{otherwise} \end{cases}$$

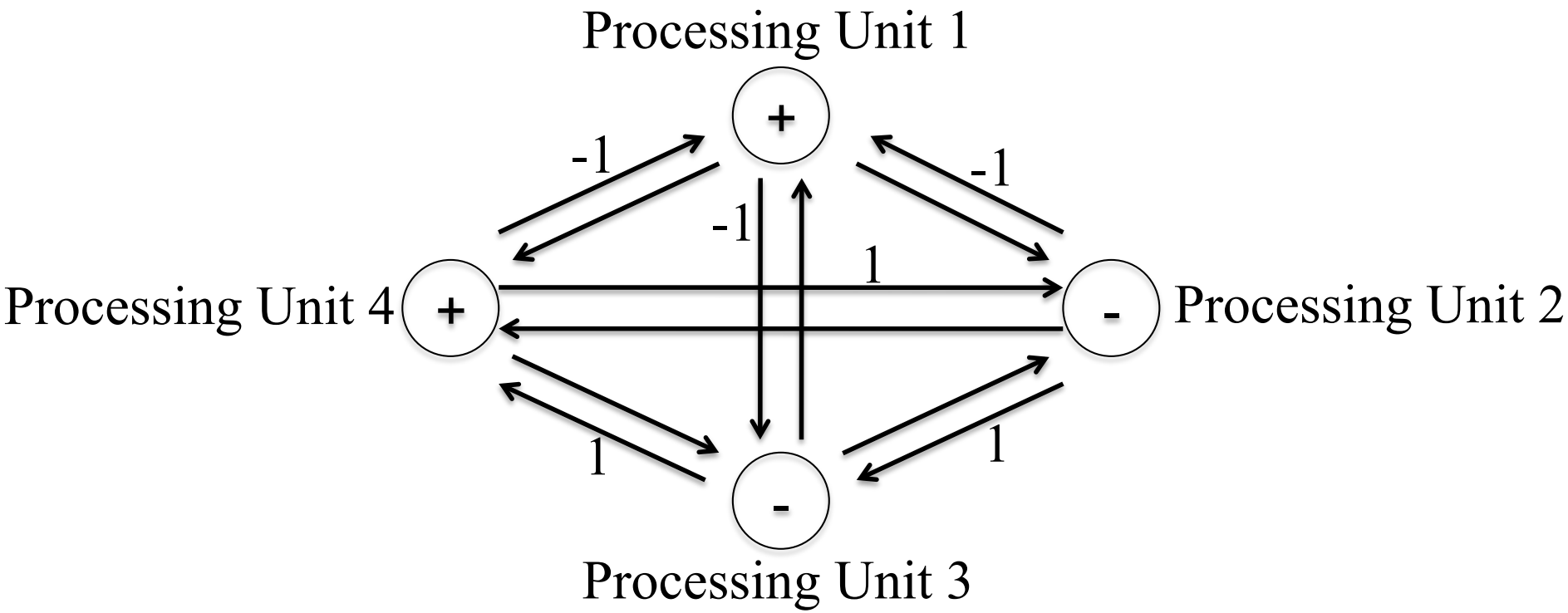
Binary Hopfield Nets

Updating Procedure

- Evaluate the sum of the weighted inputs
- Set state 1 if the sum is greater or equal 0 and 0 if sum is lower 0
- Previous state is not taken into account

Binary Hopfield Nets
Example

- Example on blackboard



Binary Hopfield Nets
Order of Updating

- Could be sequentially
- Random order (Hopfield networks)
  - Same average update rate
  - Advantages in implementation
  - Advantages in function (equiprobable stable states)
- Randomized asynchronous updating is a closer match to the biological neuronal nets

Binary Hopfield Nets

Energy Function

- Assign a numerical value to each possible state of the system (**Lyapunov Function**)
- Corresponds to the “energy” of the net
- Energy Function:

$$E = -\frac{1}{2}\sum_{j}\sum_{\substack{i \\ i \neq j}} u_i u_j T_{ji}$$

- Objective function that is optimized by the net

Binary Hopfield Nets

Proof of Convergence

- It can be shown that each updating step leads to lower or same energy in the net
- Simple case: No changes in state; Energy stays the same
- Only one unit j is updated at a time
    - Energy changes only for unit j

$$E_j = -\frac{1}{2}\sum_{\substack{i \\ i \neq j}} u_i u_j T_{ji}$$

Binary Hopfield Nets

Proof of Convergence

- Given a change in state, the difference in E is

$$\Delta E_j = E_{j_{new}} - E_{j_{old}} = -\frac{1}{2}\Delta u_j \sum_{\substack{i \\ i \neq j}} u_i T_{ji}, \quad \Delta u_j = u_{j_{new}} - u_{j_{old}}$$

- Change from 0 to 1:

$$\Delta u_j = 1, \quad \sum T_{ji} u_i \geq 0 \;\Rightarrow\; \Delta E_j \leq 0$$

- Change from 1 to 0:

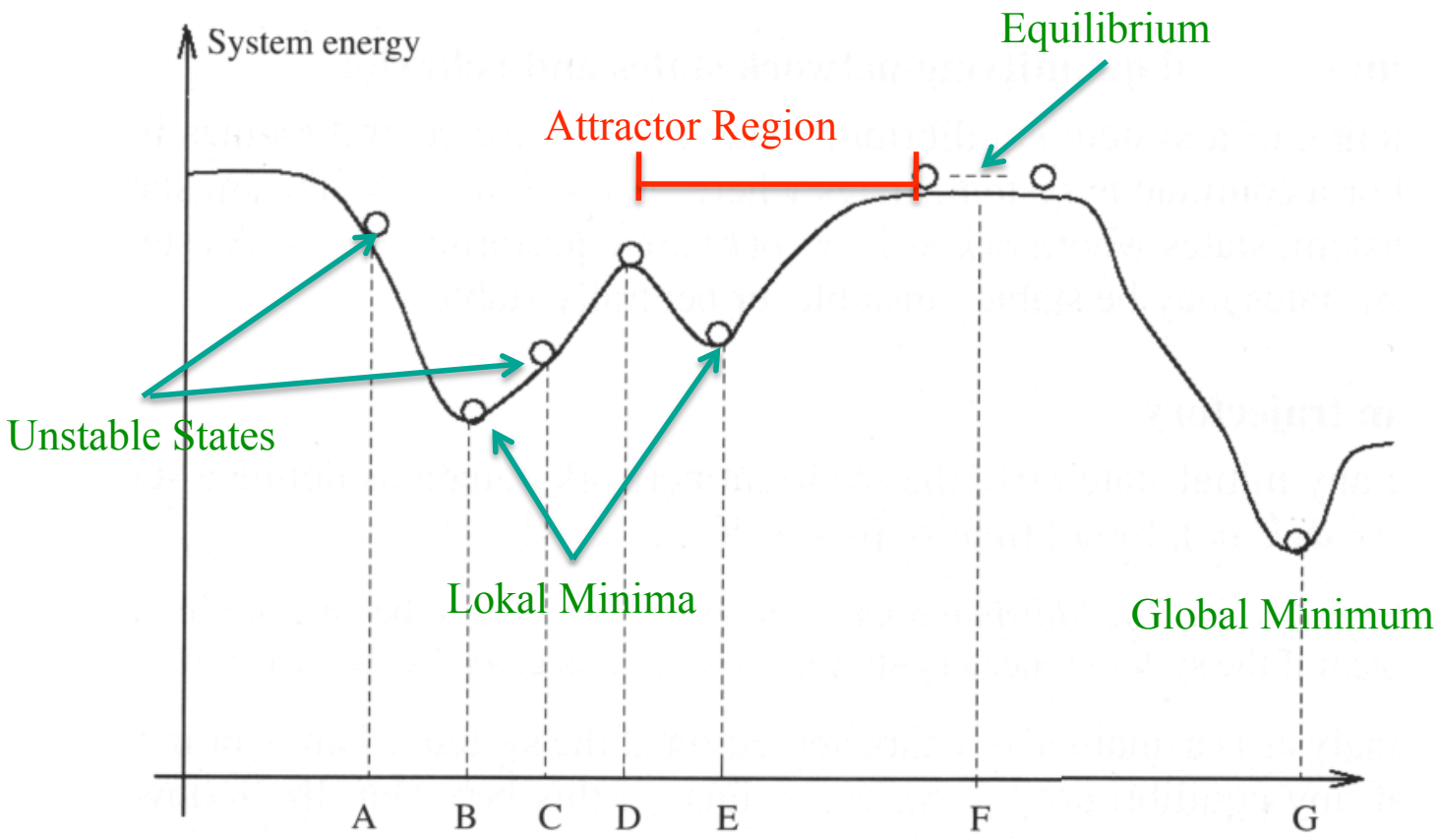
$$\Delta u_j = -1, \quad \sum T_{ji} u_i < 0 \;\Rightarrow\; \Delta E_j < 0$$

Binary Hopfield Nets
Graphical Interpretation

- Stable states are minima of the energy function
  - Can be global or local minima
- Analogous to finding a minimum in a mountainous terrain
- Depending on the starting point the state “rolls down” to a “valley”
- The landscape has as many dimensions as there are processing units in the network

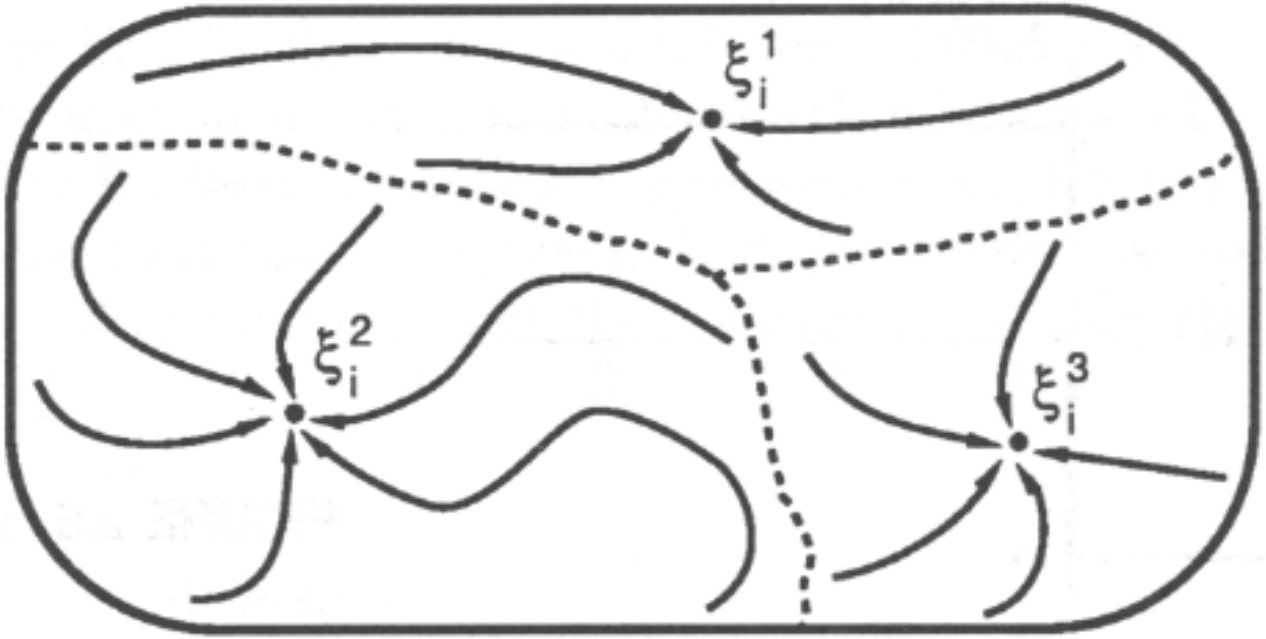
Binary Hopfield Nets

Energy Function



Binary Hopfield Nets

Stable States as Attractors



ξ's: Stable states

Binary Hopfield Nets

Application: Associative Memory

- Original purpose
- Basic Scheme:
  - A “memory” is represented by a state vector
  - Each memory vector is a stable state
  - When starting at an initial state the net converges to the most similar/most accessible memory state

Associative Memory
Example

- Storing information for different persons

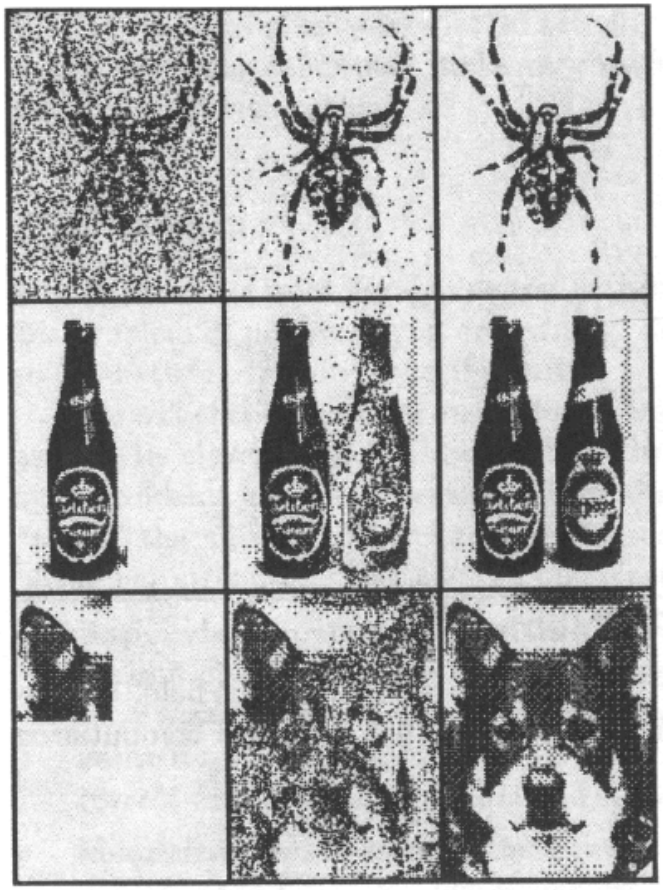
$$\vec{u}_1 = (\underbrace{+,+,-,+,-,-,-}_{\text{Name}},\underbrace{+,-,+,-,+}_{\text{Color (e.g. eyes)}})$$

- Idea: Get information by taking name and random values for color (unknown) as initial state
- Network will converge to the corresponding stable state representing the right color

Associative Memory

Noisy/incomplete Pattern Retrieval

- Each Pixel is represented by a processing unit
- Regular pattern is stored as a stable state
- Can be retrieved by taking noisy/incomplete pattern as initial state
- Only if initial state is laying in the attractors region of the stable state



Associative Memory

Setting the weights

- Select **m** patterns that have to be stored
- Number of units is equal to the entries in the patterns
- For each pattern **p** there is a Vector

$$A_p = (a_{p1}, a_{p2}, \ldots, a_{pn})$$

- For all T_{ji}

$$T_{ji} = \sum_{p=1}^{m} (2a_{pi} - 1)(2a_{pj} - 1)$$

Associative Memory

Setting the weights

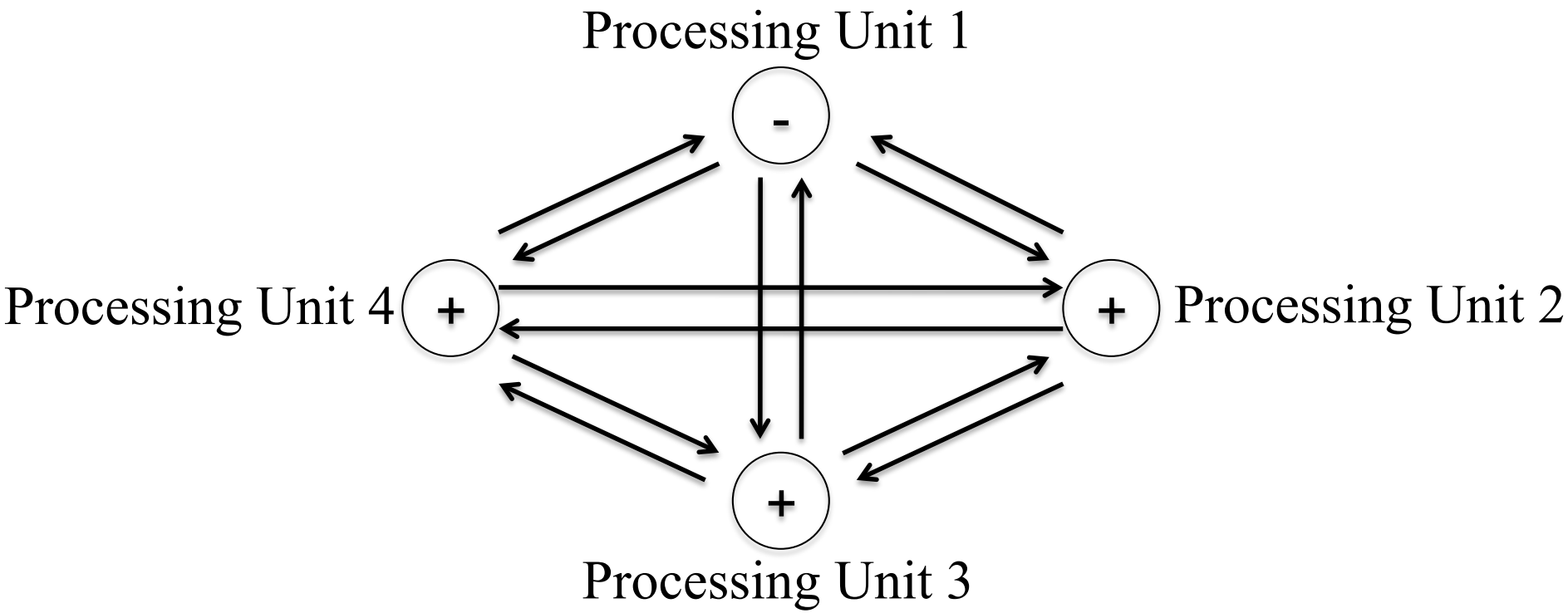
- Loop over all **m** patterns
- T_{ji} is incremented if two entries (of one pattern) j and i are the same and decremented if the entries are different
- Compare all entries of a pattern with i≠j
- 2a-1 is 1 if a is 1 and -1 if a is 0

$$T_{ji} = \sum_{p=1}^{m} (2a_{pi} - 1)(2a_{pj} - 1)$$

Associative Memory

Setting the weights

- Example on blackboard

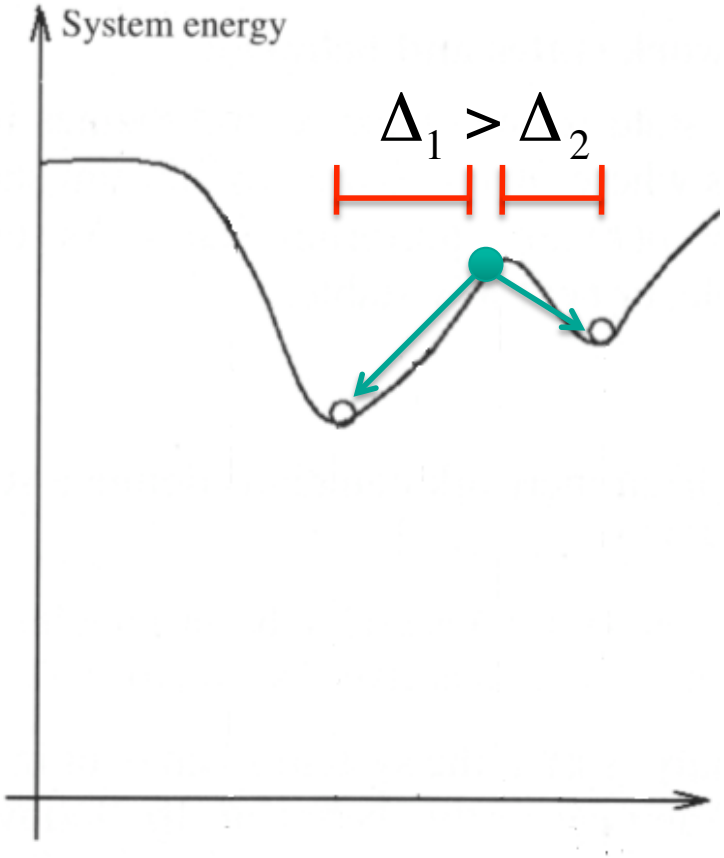


Binary Hopfield Nets
Limitations

- Found stable state (memory) is not guaranteed the most similar pattern to the input pattern
    - Not all memories are remembered with same emphasis (attractors region is not the same size)
- Spurious states can occur
- Efficiency is not good

Binary Hopfield Nets

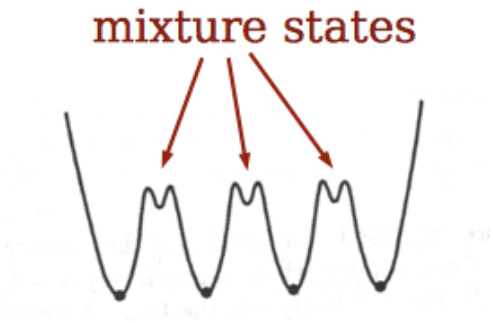
Limitations: Wrong pattern derived



Binary Hopfield Nets

Limitations: Spurious States

- Retrieval States
- Reversed States
- Mixture States: Any linear combination of an odd number of patterns



- “Spinglass” states: Stable states that are no linear combination of stored patterns (occur when too many patterns are stored)
- 3 & 4 are spurious states

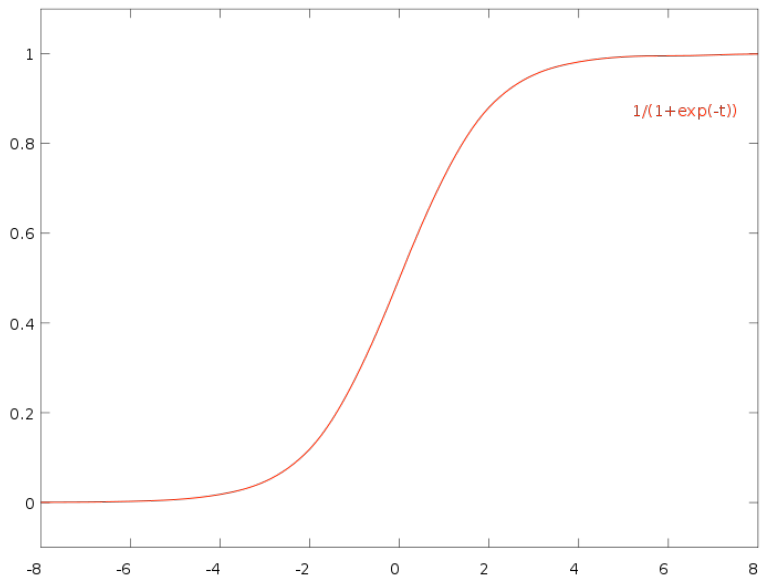
Binary Hopfield Nets
Limitations: Efficiency

- In a net of **N** units, patterns of length **N** can be stored
- Assuming uncorrelated patterns, the capacity C of a hopfield net is $C \approx 0.15N$
- Tighter bound $$\frac{N}{4\ln N} < C < \frac{N}{2\ln N}$$
- So 100 neurons can reliably store about 8 patterns

Continuous Hopfield Nets

- States can attain continuous values
- No hard threshold function, but sigmoid function as activity function (soft threshold)

$$g(x) = \frac{1}{(1 + e^{-x})}$$



Continuous Hopfield Nets

- Also update times are continuous, so differential equation
- Processing units j comply with $$C_j \frac{du_j}{dt} = \sum_i T_{ji} V_i - \frac{u_j}{R_j} + I_j$$
- Therefore the energy equation is $$E = -\frac{1}{2} \sum_j \sum_i T_{ji} u_j u_i - \sum_j u_j I_j$$
- C_j: constant > 0
- R_j: decay resistance (>0)
- I_j: external input
- V_i: output if u_i (sig. applied)

Continuous Hopfield Nets Traveling Salesman

- Matrix representation of the trip
  - Row: City
  - Col: Position on the tour
- Minimize the total distance travelled

$$\sum_{i=1}^{n} d_{X_i, X_{i+1}}$$

	1	2	3
A	0	1	0
B	0	0	1
C	1	0	0

- Minimize Energy

Continuous Hopfield Nets Traveling Salesman

- Energy Function
- Parts are restrictions on the tour
- 1: Small, if one entry per row (only one visit)
- 2: Small, if one entry per column (only one city per time)
- 3: Small, if only n entries in matrix
- 4: Proportional to the total distances of the tour
- A,B,C,D: Constants (tune for performance)

$$E = (A/2)\sum_X \sum_i \sum_{\substack{j \\ j \neq i}} V_{Xi} V_{Xj}$$
$$+ (B/2)\sum_i \sum_X \sum_{\substack{Y \\ Y \neq X}} V_{Xi} V_{Xj}$$
$$+ (C/2)(\sum_X \sum_i V_{Xi} - n^2)$$
$$+ (D/2)\sum_X \sum_i \sum_{\substack{j \\ j \neq i}} d_{XY} u_{Xi} (V_{X,i+1} + V_{Y,i-1})$$

Continuous Hopfield Nets Traveling Salesman

- Weights are set in the beginning according to

$$
\begin{aligned}
T_{Xi,Xj} = & -A\,\delta_{XY}(1-\delta_{ij}) \\
& -B\,\delta_{ij}(1-\delta_{XY}) \\
& -C \\
& -D\,d_{XY}(\delta_{j,i+1}+\delta_{j,i-1})
\end{aligned}
$$

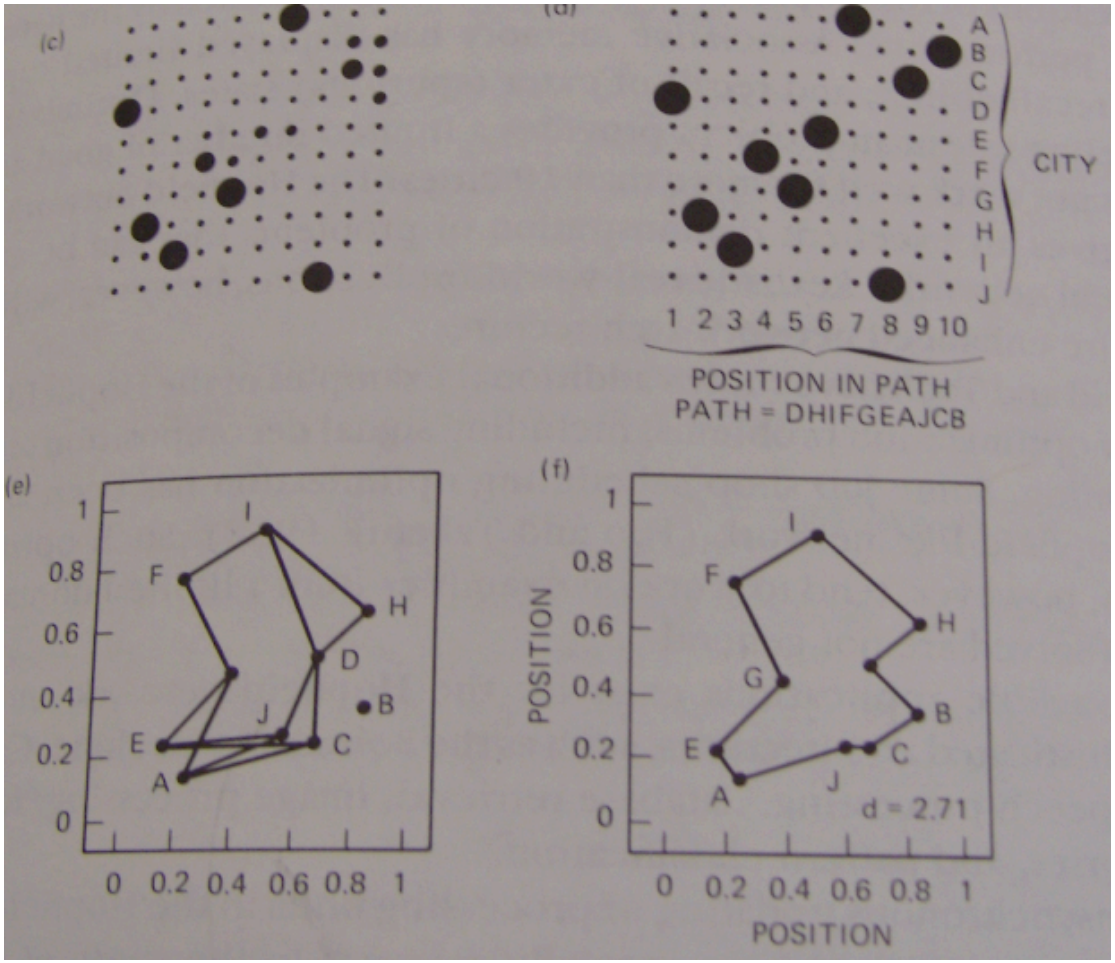
- with

$$
\delta_{ij} = \begin{cases} 1 & if\ \ i = j \\ 0 & otherwise \end{cases}
$$

Continuous Hopfield Nets Traveling Salesman

- Set random initial state
- Perform updating procedure, until the network converges
- Stable state gives the optimal route for the salesman
    - Solution is good but not optimal
    - Does not work well for more than 10 cities

Continuous Hopfield Nets Traveling Salesman



Continuous Hopfield Nets

Pro's and Con's

- Asynchronous updating is more related to the biological neurons behavior
  - Can also be helpful in designing fast hardware implementations (parallelize)
- Limited performance
  - spurious states
  - Complex computation
- Only capable of optimization in specific domains

Boltzmann Machines

- Named after Boltzmann Distribution in statistical mechanics (used for the probability of a state)
- Basic structure similar to (binary) Hopfield Nets, but **stochastic** processing units
- Hidden units
- Visible units can be divided into in- and output
- Not fully interconnected
    - No direct connections between in- and output
- Weights are still symmetric
- Still no connections to the unit itself

Boltzmann Machines

Idea: Simulated Annealing

- Define some cost function C we want to minimize
- Try to make moves that lower C
- But allow moves that raise C some probability that depends on a “temperature” parameter τ.
- Start out at high τ; “anneal” by slowly lowering τ.
- Can escape from local minima!

Boltzmann Machines

Simulated Annealing in BM

- Energy gap:

$$\Delta E_i = E(u_i = 1) - E(u_i = 0)$$
$$= -\sum_j u_j T_{ji} = -net_i$$

- Energy gap is the change in E when u_i turns on
- In Hopfield nets the change in state is only done when energy is lowered

Boltzmann Machines

Stochastic Units

- Stochastic selection of state 1 or 0

$$P(u_j = 1) = g(x_j)$$

$$P(u_j = 0) = 1 - g(x_j)$$

- $g(x_j)$ is the Boltzmann Distribution

$$g(x) = \frac{1}{1 + e^{-2\beta x}}, \quad \beta = 1/\tau$$

- x_j is still the sum of weighted inputs from other units

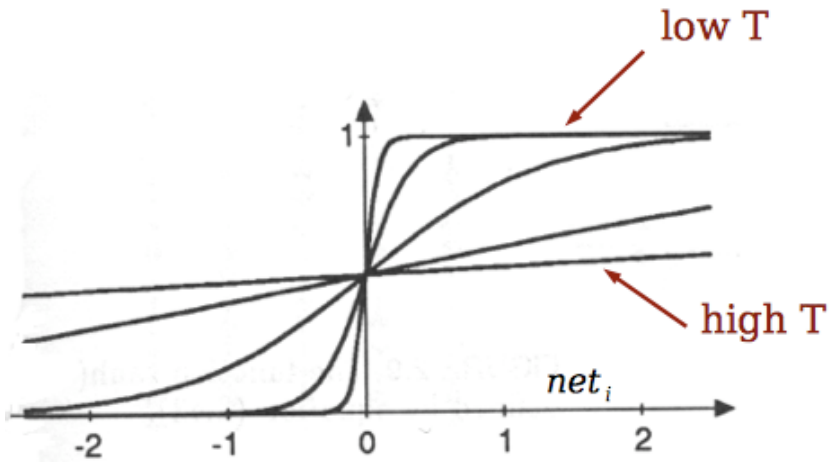
$$x_j = \sum_i u_i T_{ji}$$

Boltzmann Machines
Stochastic State Change

- In Hopfield nets states change deterministic
- Probability of changing states in BM

$$P(u_i \rightarrow -u_i) = \frac{1}{1+e^{\Delta E_i \beta}} = \frac{1}{1+e^{-net_i \beta}}$$

- Use randomness to jump out of local minima



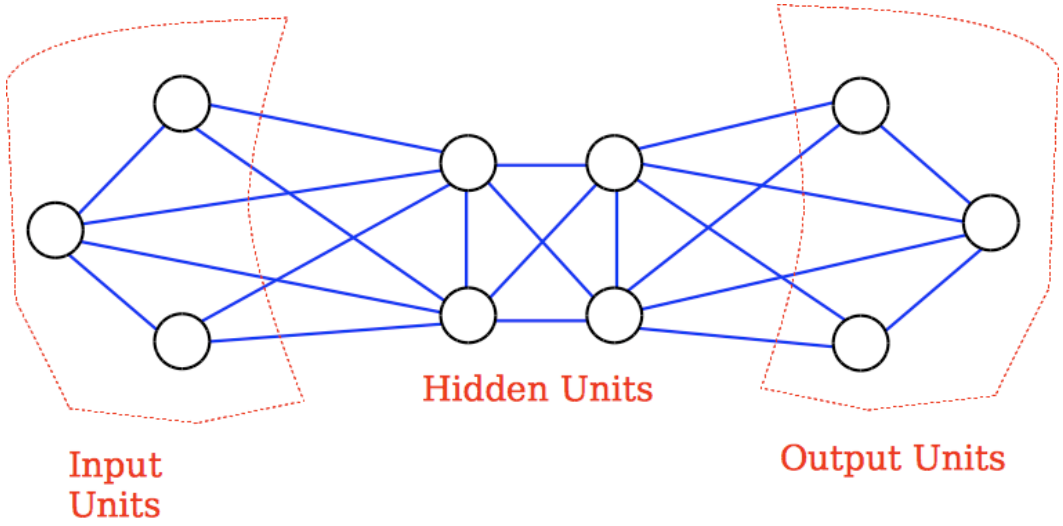
Boltzmann Machines
Stochastic Search

- Start with high temperature
  - $P[u_j \rightarrow -u_j]$ is close to 0.5. Units fluctuate a lot.
- Gradually cool to lower temperatures.
  - Units fluctuate less as P moves closer to 1 or 0.
- At zero temperature, we have a Hopfield net
- Annealing schedule:

$$\tau_{t+1} = 0.9\tau_t$$

Boltzmann Machines

Structure



- Input is set and fixed (clamped)
- Annealing is done
- Answer is presented at the output
- Hidden units add extra representational power

Boltzmann Machines
Learning Algorithm

1. Clamp inputs, anneal, measure $<u_i,u_j>^+$
2. Unclamp inputs, anneal, measure $<u_i,u_j>^-$
3. Weight update (gradient descent):

$$\Delta T_{ji} = \eta\left[\left\langle u_i,u_j\right\rangle^+ - \left\langle u_i,u_j\right\rangle^-\right]$$

- <*> := Average
- <*,*> := Correlation

Boltzmann Machines
Stochastic Search

- A Boltzmann machine with enough hidden units can compute any computable function.
- But annealing may have to be very slow.
- Mean field approximation to Boltzmann machine:
  - Replace $\vec{u}$ by $\langle \vec{u} \rangle$

- Faster than regular Boltzmann since we don't have to wait a long time to reach equilibrium state.
- But not as good as avoiding local minima.